По мотивам лекций Е. А. Соколова (ФКН ВШЭ)

1 Общая постановка задачи

Имеется пространство объектов $\mathbb{X} \subset \mathbb{R}^d$ и множество ответов $\mathbb{Y}$, подразумевается наличие (неизвестной нам) зависимости (функции) $y : \mathbb{X} \to \mathbb{Y}$.

Имеется множество $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_\ell$ объектов (обучающая выборка), для которых известны ответы

$$y(\overline{x}_i) = y_i.$$

Требуется найти алгоритм

$$a : \mathbb{X} \to \mathbb{Y},$$

который "хорошо приближается"к зависимости y на всем $\mathbb{X}$.

2 Линейные модели

На сегодняшней лекции мы рассмотрим линейные регрессионные модели. Такие модели ищут функцию a среди линейных функций и сводятся к суммированию значений признаков с некоторыми весами:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j.$$

Параметрами модели являются *веса* или *коэффициенты* w_j. Вес w_0 также называется свободным коэффициентом или *сдвигом* (bias).

Заметим, что сумма в формуле является скалярным произведением вектора признаков на вектор весов.

Запишем линейную модель в более компактном виде:

$$a(x) = \langle w, x \rangle,$$

где $w = (w_0, w_1, \ldots, w_d)^t$ — вектор весов, $x = (1, x_1, \ldots, x_d)^t$ – вектор признаков с введенным формальным признаком $x_0 = 1$ (всегда равным единице).

Линейные модели достаточно быстро и легко обучаются, и поэтому популярны при работе с большими объёмами данных. Также у них мало параметров, благодаря чему удаётся контролировать риск переобучения и использовать их для работы с зашумлёнными данными и с небольшими выборками.

3 Измерение ошибки в задачах регрессии

Чтобы обучать регрессионные модели, нужно определиться, как именно измеряется качество предсказаний.

Будем обозначать через y значение целевой переменной, через a — прогноз модели. Рассмотрим несколько способов оценить отклонение $L(y, a)$ прогноза от истинного ответа.

MSE и R^2. Основной способ измерить отклонение — посчитать квадрат разности:

$$L(y, a) = (a - y)^2$$

Благодаря своей дифференцируемости эта функция наиболее часто используется в задачах регрессии. Основанный на ней функционал называется среднеквадратичным отклонением (mean squared error, MSE):

$$\text{MSE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2 .$$

****

Отметим, что величина среднеквадратичного отклонения плохо интерпретируется, поскольку не сохраняет единицы измерения — так, если мы предсказываем цену в рублях, то MSE будет измеряться в квадратах рублей. Чтобы избежать этого, используют корень из среднеквадратичной ошибки (root mean squared error, RMSE):

$$\text{RMSE}(a, X) = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2}.$$

Среднеквадратичная ошибка подходит для сравнения двух моделей или для контроля качества во время обучения, но не позволяет сделать выводы о том, насколько хорошо данная модель решает задачу. Например, $\text{MSE} = 10$ является очень плохим показателем, если целевая переменная принимает значения от 0 до 1, и очень хорошим, если целевая переменная лежит в интервале $(10000, 100000)$. В таких ситуациях вместо среднеквадратичной ошибки полезно использовать *коэффициент детерминации* (или коэффициент R^2):

$$R^2(a, X) = 1 - \frac{\sum_{i=1}^{\ell} (a(x_i) - y_i)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})^2},$$

где $\bar{y} = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$ — среднее значение целевой переменной. Коэффициент детерминации измеряет долю дисперсии, объяснённую моделью, в общей дисперсии целевой переменной. Фактически, данная мера качества — это нормированная среднеквадратичная ошибка. Если она близка к единице, то модель хорошо объясняет данные, если же она близка к нулю, то прогнозы сопоставимы по качеству с константным предсказанием.

MAE. Заменим квадрат отклонения на модуль:

$$L(y, a) = |a - y|$$

Соответствующий функционал называется средним абсолютным отклонением (mean absolute error, MAE):

$$\text{MAE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i| .$$

Модуль отклонения не является дифференцируемым, но при этом менее чувствителен к выбросам. Квадрат отклонения, по сути, делает особый акцент на объектах с сильной ошибкой, и метод обучения будет в первую очередь стараться уменьшить отклонения на таких объектах. Если же эти объекты являются выбросами (то есть значение целевой переменной на них либо ошибочно, либо относится к другому распределению и должно быть проигнорировано), то такая расстановка акцентов приведёт к плохому качеству модели. Модуль отклонения в этом смысле гораздо более терпим к сильным ошибкам.

MSLE. Перейдём теперь к логарифмам ответов и прогнозов:

$$L(y, a) = (\log(a + 1) - \log(y + 1))^2$$

Соответствующий функционал называется среднеквадратичной логарифмической ошибкой (mean squared logarithmic error, MSLE). Данная метрика подходит для задач с неотрицательной целевой переменной. За счёт логарифмирования ответов и прогнозов мы скорее штрафуем за отклонения в порядке величин, чем за отклонения в их значениях. Также следует помнить, что логарифм не является симметричной функцией, и поэтому данная функция потерь штрафует заниженные прогнозы сильнее, чем завышенные.

MAPE и SMAPE. В задачах прогнозирования обычно измеряется относительная ошибка:

$$L(y, a) = \left|\frac{y - a}{y}\right|$$

Соответствующий функционал называется средней абсолютной процентной ошибкой (mean absolute percentage error, MAPE). Данный функционал часто используется в задачах прогнозирования. Также используется его симметричная модификация (symmetric mean absolute percentage error, SMAPE):

$$L(y, a) = \frac{|y - a|}{(|y| + |a|)/2}$$

****

4 Обучение линейной регрессии

Чаще всего линейная регрессия обучается с использованием среднеквадратичной ошибки. В этом случае получаем задачу оптимизации (считаем, что среди признаков есть константный, и поэтому свободный коэффициент не нужен):

$$\frac{1}{\ell}\sum_{i=1}^{\ell}(\langle w, x_i\rangle - y_i)^2 \to \min_w$$

Эту задачу можно переписать в матричном виде.

Обозначим через X — матрицу «объектов-признаков»

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{\ell 1} & x_{\ell 2} & \dots & x_{\ell d} \end{pmatrix}$$

Если y — вектор ответов, а w — вектор параметров, то приходим к виду

$$\frac{1}{\ell} \|Xw - y\|^2 \to \min_w,$$

где используется обычная L_2-норма. Если продифференцировать данный функционал по вектору w, приравнять к нулю и решить уравнение, то получим явную формулу для решения:

$$w = (X^T X)^{-1} X^T y.$$

Вывод формулы

Введем обозначения для формального матричного дифференцирования.

Производная векторной функции

$$f : \mathbb{R} \to \mathbb{R}^n$$

по скаляру – вектор той же размерности, состоящий из производных соответствующих элементов

$$\frac{df(t)}{dt} = \left(\frac{\partial f_1(t)}{\partial t}, \frac{\partial f_2(t)}{\partial t}, \dots, \frac{\partial f_n(t)}{\partial t} \right)^t.$$

Производная скалярной функции $f : \mathbb{R}^n \to \mathbb{R}$ от векторного аргумента x — вектор той же размерности, что и x, равный

$$\frac{df(x)}{dx} = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots, \frac{\partial f(x)}{\partial x_n} \right)^t.$$

Производная векторной функции $f : \mathbb{R}^n \to \mathbb{R}^m$ размерности m по векторному аргументу x размерности n – матрица размером $m \times n$, в которой первая строка состоит из частных производных элемента f_1 по элементам вектора x и т.д.:

$$\frac{df(x)}{dx} = \begin{pmatrix} \dfrac{\partial f_1(x)}{\partial x_1} & \dfrac{\partial f_2(x)}{\partial x_1} & \dots & \dfrac{\partial f_n(x)}{\partial x_1} \\ \dots & \dots & \dots & \dots \\ \dfrac{\partial f_1(x)}{\partial x_m} & \dfrac{\partial f_2(x)}{\partial x_m} & \dots & \dfrac{\partial f_n(x)}{\partial x_m} \end{pmatrix}$$

Нам понадобятся две формулы (доказать!).

$$\frac{d\langle x, w \rangle}{dx} = \frac{d(x^t \cdot w)}{dx} = w.$$

Если A – квадратная матрица, то

$$\frac{d(x^t A x)}{dx} = (A + A^t)x.$$

Если A – симметричная квадратная матрица, то формула упроститься до

$$\frac{d(x^tAx)}{dx} = 2Ax.$$

$$\|Xw - y\|^2 = (Xw - y)^t(Xw - y) = w^tX^tXw - w^tX^ty - y^tXw + y^ty.$$

$$\frac{\|Xw - y\|^2}{dw} = 2X^tXw - 2X^ty = 0.$$

$$w = (X^TX)^{-1}X^Ty.$$

Комментарии

Безусловно, наличие явной формулы для оптимального вектора весов — это большое преимущество линейной регрессии с квадратичным функционалом. Но данная формула не всегда применима по ряду причин:

- Обращение матрицы — сложная операция с кубической сложностью от количества признаков. Если в выборке тысячи признаков, то вычисления могут стать слишком трудоёмкими. Решить эту проблему можно путём использования численных методов оптимизации.
- Матрица X^TX может быть вырожденной или плохо обусловленной. В этом случае обращение либо невозможно, либо может привести к неустойчивым результатам. Проблема решается с помощью регуляризации, речь о которой пойдёт ниже.

Следует понимать, что аналитические формулы для решения довольно редки в машинном обучении. Если мы заменим MSE на другой функционал, то найти такую формулу, скорее всего, не получится. Желательно разработать общий подход, в рамках которого можно обучать модель для широкого класса функционалов. Такой подход действительно есть для дифференцируемых функций — обсудим его подробнее.

5 Градиентный спуск и оценивание градиента

Оптимизационные задачи можно решать итерационно с помощью градиентных методов (или же методов, использующих как градиент, так и информацию о производных более высокого порядка).

Градиентом функции $f : \mathbb{R}^d \to \mathbb{R}$ называется вектор его частных производных:

$$\nabla f(x_1, \ldots, x_d) = \left(\frac{\partial f}{\partial x_j}\right)_{j=1}^{d}.$$

Известно, что градиент является направлением наискорейшего роста функции, а антиградиент (т.е. $-\nabla f$) — направлением наискорейшего убывания. Это ключевое свойство градиента, обосновывающее его использование в методах оптимизации.

Докажем данное утверждение. Пусть $v \in \mathbb{R}^d$ — произвольный вектор, лежащий на единичной сфере: $\|v\| = 1$. Пусть $x_0 \in \mathbb{R}^d$ — фиксированная точка пространства. Скорость роста функции в точке x_0 вдоль вектора v характеризуется производной по направлению $\frac{\partial f}{\partial v}$:

$$\frac{\partial f}{\partial v} = \frac{d}{dt} f(x_{0,1} + tv_1, \ldots, x_{0,d} + tv_d)|_{t=0}.$$

Из курса математического анализа известно, что данную производную сложной функции можно переписать следующим образом:

$$\frac{\partial f}{\partial v} = \sum_{j=1}^{d} \frac{\partial f}{\partial x_j} \frac{d}{dt} (x_{0,j} + tv_j) = \sum_{j=1}^{d} \frac{\partial f}{\partial x_j} v_j = \langle \nabla f, v \rangle.$$

Распишем скалярное произведение:

$$\langle \nabla f, v \rangle = \|\nabla f\| \|v\| \cos \varphi = \|\nabla f\| \cos \varphi,$$

где φ — угол между градиентом и вектором v. Таким образом, производная по направлению будет максимальной, если угол между градиентом и направлением равен нулю, и минимальной, если угол равен 180 градусам. Иными словами, производная по направлению максимальна вдоль градиента и минимальна вдоль антиградиента.

У градиента есть ещё одно свойство, которое пригодится нам при попытках визуализировать процесс оптимизации, — он ортогонален линиям уровня. Докажем это. Пусть x_0 — некоторая точка, $S(x_0) = \{x \in \mathbb{R}^d \,|\, f(x) = f(x_0)\}$ — соответствующая линия уровня. Разложим функцию в ряд Тейлора на этой линии в окрестности x_0:

$$f(x_0 + \varepsilon) = f(x_0) + \langle \nabla f, \varepsilon \rangle + o(\|\varepsilon\|),$$

где $x_0 + \varepsilon \in S(x_0)$. Поскольку $f(x_0 + \varepsilon) = f(x_0)$ (как-никак, это линия уровня), получим

$$\langle \nabla f, \varepsilon \rangle = o(\|\varepsilon\|).$$

Поделим обе части на $\|\varepsilon\|$:

$$\left\langle \nabla f, \frac{\varepsilon}{\|\varepsilon\|} \right\rangle = o(1).$$

Устремим $\|\varepsilon\|$ к нулю. При этом вектор $\frac{\varepsilon}{\|\varepsilon\|}$ будет стремится к касательной к линии уровня в точке x_0. В пределе получим, что градиент ортогонален этой касательной.

§5.1 Градиентный спуск

Основное свойство антиградиента — он указывает в сторону наискорейшего убывания функции в данной точке. Соответственно, будет логично стартовать из некоторой точки, сдвинуться в сторону антиградиента, пересчитать антиградиент и снова сдвинуться в его сторону и т.д. Запишем это более формально. Пусть $w^{(0)}$ — начальный набор параметров (например, нулевой или сгенерированный из некоторого

случайного распределения). Тогда градиентный спуск состоит в повторении следующих шагов до сходимости:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla Q(w^{(k-1)}). \tag{5.1}$$

Здесь под $Q(w)$ понимается значение функционала ошибки для набора параметров w.

Через η_k обозначается длина шага, которая нужна для контроля скорости движения. Можно делать её константной: $\eta_k = c$. При этом если длина шага слишком большая, то есть риск постоянно «перепрыгивать» через точку минимума, а если шаг слишком маленький, то движение к минимуму может занять слишком много итераций. Иногда длину шага монотонно уменьшают по мере движения — например, по простой формуле

$$\eta_k = \frac{1}{k}.$$

В пакете `vowpal wabbit`, реализующем настройку и применение линейных моделей, используется более сложная формула для шага в градиентном спуске:

$$\eta_k = \lambda \left(\frac{s_0}{s_0 + k} \right)^p,$$

где λ, s_0 и p — параметры (мы опустили в формуле множитель, зависящий от номера прохода по выборке). На практике достаточно настроить параметр λ, а остальным присвоить разумные значения по умолчанию: $s_0 = 1$, $p = 0.5$, $d = 1$.

Останавливать итерационный процесс можно, например, при близости градиента к нулю или при слишком малом изменении вектора весов на последней итерации.

Если функционал $Q(w)$ выпуклый, гладкий и имеет минимум w^*, то имеет место следующая оценка сходимости:

$$Q(w^{(k)}) - Q(w^*) = O(1/k).$$

§5.2 Оценивание градиента

Как правило, в задачах машинного обучения функционал $Q(w)$ представим в виде суммы ℓ функций:

$$Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} q_i(w).$$

В таком виде, например, записан функционал в задаче (**??**), где отдельные функции $q_i(w)$ соответствуют ошибкам на отдельных объектах.

Проблема метода градиентного спуска (5.1) состоит в том, что на каждом шаге необходимо вычислять градиент всей суммы (будем его называть полным градиентом):

$$\nabla_w Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} \nabla_w q_i(w).$$

Это может быть очень трудоёмко при больших размерах выборки. В то же время точное вычисление градиента может быть не так уж необходимо — как правило, мы делаем не очень большие шаги в сторону антиградиента, и наличие в нём неточностей не должно сильно сказаться на общей траектории. Опишем несколько способов оценивания полного градиента.

Оценить градиент суммы функций можно градиентом одного случайно взятого слагаемого:

$$\nabla_w Q(w) \approx \nabla_w q_{i_k}(w),$$

где i_k — случайно выбранный номер слагаемого из функционала. В этом случае мы получим метод *стохастического градиентного спуска* (stochastic gradient descent, SGD) [1]:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla q_{i_k}(w^{(k-1)}).$$

Для выпуклого и гладкого функционала может быть получена следующая оценка:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/\sqrt{k}).$$

Таким образом, метод стохастического градиента имеет менее трудоемкие итерации по сравнению с полным градиентом, но и скорость сходимости у него существенно меньше.

Отметим одно важное преимущество метода стохастического градиентного спуска. Для выполнения одного шага в данном методе требуется вычислить градиент лишь одного слагаемого — а поскольку одно слагаемое соответствует ошибке на одном объекте, то получается, что на каждом шаге необходимо держать в памяти всего один объект из выборки. Данное наблюдение позволяет обучать линейные модели на очень больших выборках: можно считывать объекты с диска по одному, и по каждому делать один шаг метода SGD.

Можно повысить точность оценки градиента, используя несколько слагаемых вместо одного:

$$\nabla_w Q(w) \approx \frac{1}{n} \sum_{j=1}^{n} \nabla_w q_{i_{kj}}(w),$$

где i_{kj} — случайно выбранные номера слагаемых из функционала (j пробегает значения от 1 до n), а n — параметр метода, размер пачки объектов для одного градиентного шага. С такой оценкой мы получим метод mini-batch gradient descent, который часто используется для обучения дифференцируемых моделей.

В 2013 году был предложен метод *среднего стохастического градиента* (stochastic average gradient) [2], который в некотором смысле сочетает низкую сложность итераций стохастического градиентного спуска и высокую скорость сходимости полного градиентного спуска. В начале работы в нём выбирается первое приближение w^0, и инициализируются вспомогательные переменные z_i^0, соответствующие градиентам слагаемых функционала:

$$z_i^{(0)} = \nabla q_i(w^{(0)}), \qquad i = 1, \ldots, \ell.$$

На k-й итерации выбирается случайное слагаемое i_k и обновляются вспомогательные переменные:

$$z_i^{(k)} = \begin{cases} \nabla q_i(w^{(k-1)}), & \text{если } i = i_k; \\ z_i^{(k-1)} & \text{иначе.} \end{cases}$$

Иными словами, пересчитывается один из градиентов слагаемых. Оценка градиента вычисляется как среднее вспомогательных переменных — то есть мы используем все слагаемые, как в полном градиенте, но при этом почти все слагаемые берутся с предыдущих шагов, а не пересчитываются:

$$\nabla_w Q(w) \approx \frac{1}{\ell} \sum_{i=1}^{\ell} z_i^{(k)}.$$

Наконец, делается градиентный шаг:

$$w^{(k)} = w^{(k-1)} - \eta_k \frac{1}{\ell} \sum_{i=1}^{\ell} z_i^{(k)}.$$

Данный метод имеет такой же порядок сходимости для выпуклых и гладких функционалов, как и обычный градиентный спуск:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/k).$$

Существует множество других способов получения оценки градиента. Например, это можно делать без вычисления каких-либо градиентов вообще [3] — достаточно взять случайный вектор u на единичной сфере и домножить его на значение функции в данном направлении:

$$\nabla_w Q(w) = Q(w + \delta u)u.$$

Можно показать, что данная оценка является несмещённой для сглаженной версии функционала Q.

В задаче оценивания градиента можно зайти ещё дальше. Если вычислять градиенты $\nabla_w q_i(w)$ сложно, то можно *обучить модель*, которая будет выдавать оценку градиента на основе текущих значений параметров. Этот подход был предложен для обучения глубинных нейронных сетей [4].

§5.3 Модификации градиентного спуска

С помощью оценок градиента можно уменьшать сложность одного шага градиентного спуска, но при этом сама идея метода не меняется — мы движемся в сторону наискорейшего убывания функционала. Конечно, такой подход не идеален, и можно по-разному его улучшать, устраняя те или иные его проблемы. Мы разберём два примера таких модификаций — одна будет направлена на борьбу с осцилляциями, а вторая позволит автоматически подбирать длину шага.

Метод импульса (momentum). Может оказаться, что направление антиградиента сильно меняется от шага к шагу. Например, если линии уровня функционала сильно вытянуты, то из-за ортогональности градиента линиям уровня он будет менять направление на почти противоположное на каждом шаге. Такие осцилляции будут вносить сильный шум в движение, и процесс оптимизации займёт много итераций. Чтобы избежать этого, можно усреднять векторы антиградиента с нескольких предыдущих шагов — в этом случае шум уменьшится, и такой средний вектор будет указывать в сторону общего направления движения. Введём для этого вектор инерции:

$$h_0 = 0;$$
$$h_k = \alpha h_{k-1} + \eta_k \nabla_w Q(w^{(k-1)}).$$

Здесь α — параметр метода, определяющей скорость затухания градиентов с предыдущих шагов. Разумеется, вместо вектора градиента может быть использована его аппроксимация. Чтобы сделать шаг градиентного спуска, просто сдвинем предыдущую точку на вектор инерции:

$$w^{(k)} = w^{(k-1)} - h_k.$$

Заметим, что если по какой-то координате градиент постоянно меняет знак, то в результате усреднения градиентов в векторе инерции эта координата окажется близкой к нулю. Если же по координате знак градиента всегда одинаковый, то величина соответствующей координаты в векторе инерции будет большой, и мы будем делать большие шаги в соответствующем направлении.

AdaGrad и RMSprop. Градиентный спуск очень чувствителен к выбору длины шага. Если шаг большой, то есть риск, что мы будем «перескакивать» через точку минимума; если же шаг маленький, то для нахождения минимума потребуется много итераций. При этом нет способов заранее определить правильный размер шага — к тому же, схемы с постепенным уменьшением шага по мере итераций могут тоже плохо работать.

В методе AdaGrad предлагается сделать свою длину шага для каждой компоненты вектора параметров. При этом шаг будет тем меньше, чем более длинные шаги мы делали на предыдущих итерациях:

$$G_{kj} = G_{k-1,j} + (\nabla_w Q(w^{(k-1)}))_j^2;$$
$$w_j^{(k)} = w_j^{(k-1)} - \frac{\eta_t}{\sqrt{G_{kj} + \varepsilon}} (\nabla_w Q(w^{(k-1)}))_j.$$

Здесь ε — небольшая константа, которая предотвращает деление на ноль. В данном методе можно зафксировать длину шага (например, $\eta_k = 0.01$) и не подбирать её в процессе обучения. Отметим, что данный метод подходит для разреженных задач, в которых у каждого объекта большинство признаков равны нулю. Для признаков, у которых ненулевые значения встречаются редко, будут делаться большие шаги; если же какой-то признак часто является ненулевым, то шаги по нему будут небольшими.

У метода AdaGrad есть большой недостаток: переменная G_{kj} монотонно растёт, из-за чего шаги становятся всё медленнее и могут остановиться ещё до того,

как достигнут минимум функционала. Проблема решается в методе RMSprop, где используется экспоненциальное затухание градиентов:

$$G_{kj} = \alpha G_{k-1,j} + (1-\alpha)(\nabla_w Q(w^{(k-1)}))_j^2.$$

В этом случае размер шага по координате зависит в основном от того, насколько быстро мы двигались по ней на последних итерациях.

6 Области применимости линейных моделей

Сложно представить себе ситуацию, в которой мы берём данные, обучаем линейную модель и получаем хорошее качество работы. В линейной модели предполагается конкретный вид зависимости — а именно, что каждый признак линейно влияет на целевую переменную, и что целевая переменная не зависит от каких-либо комбинаций признаков. Вряд ли это будет выполнено по умолчанию, поэтому обычно данные требуют специальной подготовки, чтобы линейные модели оказались адекватными задаче. Приведём несколько примеров.

Категориальные признаки. Представим себе задачу определения стоимости квартиры по её характеристикам. Одним из важных признаков является район, в котором находится квартира. Этот признак является категориальным — его значения нельзя сравнивать между собой на больше/меньше, их нельзя складывать или вычитать. Непосредственно такие признаки нельзя использовать в линейных моделях, но есть достаточно распространённый способ их преобразования.

Допустим, категориальный признак $f_j(x)$ принимает значения из множества $C = \{c_1, \ldots, c_m\}$. Заменим его на m бинарных признаков $b_1(x), \ldots, b_m(x)$, каждый из которых является индикатором одного из возможных категориальных значений:

$$b_i(x) = [f_j(x) = c_i].$$

Такой подход называется one-hot кодированием.

Отметим, что признаки $b_1(x), \ldots, b_m(x)$ являются линейно зависимыми: для любого объекта выполнено

$$b_1(x) + \cdots + b_m(x) = 1.$$

Чтобы избежать этого, можно выбрасывать один из бинарных признаков. Впрочем, такое решение имеет и недостатки — например, если на тестовой выборке появится новая категория, то её как раз можно закодировать с помощью нулевых бинарных признаков; при удалении одного из них это потеряет смысл.

Вернёмся к задаче про стоимость квартиры. Если мы применим линейную модель к данным после one-hot кодирования признака о районе (допустим, это $f(x)$), то получится такая формула:

$$a(x) = w_1[f(x) = c_1] + \cdots + w_m[f(x) = c_m] + \{\text{взаимодействие с другими признаками}\}.$$

Такая зависимость кажется логичной — каждый район задаёт некоторый базовый уровень стоимости (например, для района c_1 имеем базовую цену w_1), а остальные факторы корректируют его.

Работа с текстами. Перейдём к предсказанию стоимости квартиры по её текстовому описанию. Есть простой способ кодирования, который называется *мешок слов (bag of words)*.

Найдём все слова, которые есть в нашей выборке текстов, и пронумеруем их: $\{c_1, \ldots, c_m\}$. Будем кодировать текст m признаками $b_1(x), \ldots, b_m(x)$, где $b_j(x)$ равен количеству вхождений слова c_j в текст. Линейная модель над такими признаками будет иметь вид

$$a(x) = w_1 b_1(x) + \cdots + w_m b_m(x) + \ldots,$$

и такой вид тоже кажется разумным. Каждое вхождение слова c_j меняет прогноз стоимости на w_j. В самом деле, можно ожидать, что слово «престижный» скорее говорит о том, что квартира дорогая, а слово «плохой» вряд ли будут использовать при описании приличной квартиры.

Бинаризация числовых признаков. Наконец, подумаем о предсказании стоимости квартиры по расстоянию до ближайшей станции метро x_j. Может оказаться, что самые дорогие квартиры расположены где-то в 5-10 минутах ходьбы от метро, а те, что ближе или дальше, стоят не так дорого. В этом случае зависимость целевой переменной от признака не будет линейной. Чтобы сделать линейную модель подходящей, мы можем бинаризовать признак. Для этого выберем некоторую сетку точек $\{t_1, \ldots, t_m\}$. Это может быть равномерная сетка между минимальным и максимальным значением признака или, например, сетка из эмпирических квантилей. Добавим сюда точки $t_0 = -\infty$ и $t_{m+1} = +\infty$. Новые признаки зададим как

$$b_i(x) = [t_{i-1} < x_j \leqslant t_i], \quad i = 1, \ldots, m+1.$$

Линейная модель над этими признаками будет выглядеть как

$$a(x) = w_1[t_0 < x_j \leqslant t_1] + \cdots + w_{m+1}[t_m < x_j \leqslant t_{m+1}] + \ldots,$$

то есть мы найдём свой прогноз стоимости квартиры для каждого интервала расстояния до метро. Такой подход позволит учесть нелинейную зависимость между признаком и целевой переменной.

Список литературы

[1] *Robbins, H., Monro S.* (1951). A stochastic approximation method. // Annals of Mathematical Statistics, 22 (3), p. 400-407.

[2] *Schmidt, M., Le Roux, N., Bach, F.* (2013). Minimizing finite sums with the stochastic average gradient. // Arxiv.org.

[3] *Flaxman, Abraham D. and Kalai, Adam Tauman and McMahan, H. Brendan* (2005). Online Convex Optimization in the Bandit Setting: Gradient Descent Without a Gradient. // Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms.

[4] *Jaderberg, M. et. al* (2016). Decoupled Neural Interfaces using Synthetic Gradients. // Arxiv.org.